AN MIT ENTERPRISE
TECHNOLOGY REVIEW

Taking Programming to the Extreme

By Erik Sherman July 19, 2002

Illustration by Matthew Bouchard

The quest for quality software may require programmers to lose the cowboy attitude and learn to cooperate.

Mansour Raad had a big problem. His start-up firm, DiscoverCast, was developing collision-detection software for the airline industry—mistakes in the code could cost lives. But money dried up after a first round of funding, and hiring additional programmers to finish a bug-proof version was out of the question.

So Raad instigated a relatively new coding discipline called extreme programming. Instead of working alone on individual assignments, programmers paired off, one writing code while the other watched and critiqued—providing both a critical eye and critically important backup. "[My wife] had a baby, but the project had to go on," says Raad. "Because I was showing it to the other people, someone could step into my shoes." The company met deadlines without an increase in errors because of the communication.

Most programmers are no more satisfied with software quality than their customers. Now a growing number are trying to find ways to create solid software in a market that expects new products and enhancements and wants them fast. Increasingly, software companies are turning to development strategies with names like "agile development" and "extreme programming"—adrenaline-charged catchphrases well-suited to coders' self-image as rugged heroes of the information age. In reality, the trend—which has a long history—signals a rejection of prima donna programming in favor of teamwork and collaboration.

In the software industry, buggy code has long been accepted as a fact of life—albeit an aggravating and expensive one (*see "Why Software Is So Bad," TR July/August 2002*). "The problem is that in software, people place too much trust in their own ingenuity," says Michael Stiefel, a consultant who trains clients in software reliability methods. "I'd watch all these young guys in their 20s making the same mistakes we made in the '70s and '80s, and I wonder why we have to go through all this again."

To reduce errors, reformers are touting approaches to software engineering—known in the industry as "agile development"—that stress teamwork, collaboration with end users, and a flexible approach to change. To build its collision-avoidance software, Raad's company used one type of agile development, called "extreme programming," which emphasizes constant testing as well as collaboration.

"If you are able to apply [a collaborative environment] at its full power, the quality of the products will be better," says Michele Marchesi, a leading advocate of extreme programming and professor of electrical engineering at the University of Cagliari in Italy.

One of the most powerful aspects of agile programming is so-called peer programming, in which developers partner and take turns writing code and explaining its logic to the other. Pairs are temporary, periodically splitting and swapping to encourage even more interaction. New eyes examine each piece of code, creating a process of continuous review.

"Two people can bounce ideas off each other," says Frank Arkell, a chief software engineer at defense contractor General Dynamics Decision Systems, which has used collaborative techniques for two decades. To keep teams fresh, General Dynamics changes them for every project.

Another advantage to peer programming? Peer pressure. "No one wants to be the person that slacks off, so they both work harder," says Scott W. Ambler, president of Ronin International and an author and speaker on agile development.

Put to the Test

Collaborative approaches are also transforming the way software is tested. Traditionally, testing has been a two-step process. First, programmers write code based on requirements. Then a separate group tests the result. But when software includes millions of lines of code, the two-step process is analogous to designing an automobile on paper, building it, then checking to see if the design worked.

That approach just didn't work for developers like Nicholas Stamos, chief technical officer of Waltham, MA-based Phase Forward. Phase Forward develops software for running pharmaceutical clinical trials, and its products must pass the stringent requirements of the U.S. Food and Drug Administration.

Stamos says Phase Forward meets those requirements, in part because of close and constant collaboration between the company's programmers and quality assurance personnel. "You can't throw it over the transom and hope," he says. "Quality has to be built in from day one." When programmers complete a component, the quality team checks their work. Then the teams swap places for another round of bug fixes. Finally, the integrated application is tested to find problems with the interaction between components. Instead of the old two-step approach, testing becomes an interactive, ongoing process.

Some companies are trying a similarly iterative approach not just to testing, but to designing code in the first place. Cognizant Technology Solutions, a consulting firm based in Teaneck, NJ, has found that existing methods of starting with formal specifications and then writing code to satisfy them might have worked 20 or 30 years ago. But in a world that runs on the Web and needs flexibility, formal can mean stiff.

"When we tried to use the traditional processes for these newer types of projects, either the customers got really frustrated because we forced them to freeze the process or the project teams would just not follow the process," says Cognizant CEO Kumar Mahadeva. Instead, projects use a series of prototypes followed by an "industrialization" cycle, letting the firm handle sudden changes from customers in a way that has less impact on reliability than adjusting a program after it is completed. Future users provide a steady stream of feedback as the application takes shape.

But the ultimate limits on software quality may not lie in the skills of the programmers, or the strength of the development process. "This is the conundrum that we and many manufacturers face today: what is an acceptable level of quality?" says James Hymel, director of software engineering at Motorola. "If you have competitors turning out cute, cheap, but buggy [products] and the customers say that is acceptable—that is, they spend their money on them—you'd be challenged."

With methods like agile development, partnering of development and testing, and the involvement of end-users, software developers are taking steps to improve their product. But at the end of the day, the greatest constraint to quality may be the same people demanding it the loudest: the consumers.

Erik Sherman is a contributing writer.